[PSI3472-2023. Aula 7. Início.]

Transfer learning

Para treinar uma rede neural, por exemplo para classificar uma imagem de raio-X como câncer ou não-câncer, não se costuma iniciar o treino "do zero", a partir dos pesos e vieses escolhidos aleatoriamente. Costuma-se pegar uma rede já "pré-treinada" para outras tarefas e adaptá-la para o problema de classificação de câncer. Isto se chama "transfer learning" e é o que veremos nesta aula.

I. Classificação de rostos em M/F

1. Rede "tipo LeNet" treinada "do zero"

O site abaixo contém vários bancos de dados de faces humanas:

http://fei.edu.br/~cet/facedatabase.html

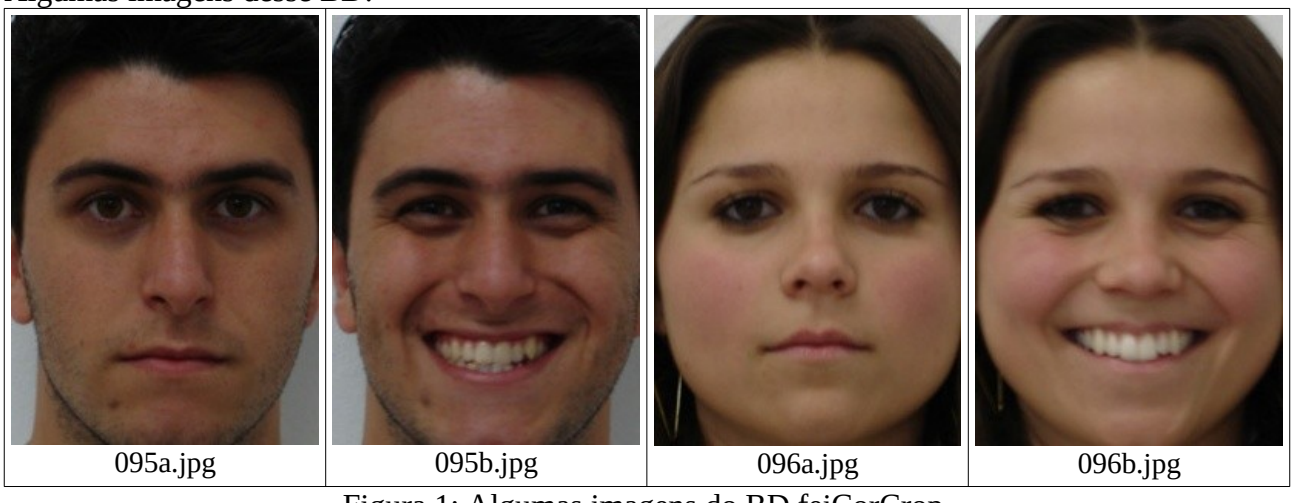
Desse site, peguei o banco de imagens com faces frontais de 200 pessoas (400 imagens coloridas com 360×260 pixels) de rostos frontais com expressão neutra (*a.jpg) e sorridente (*b.jpg), alinhadas manualmente. Dessas imagens, metade são rostos masculinos e metade são femininos. Recortei as bordas dessas imagens, para que fiquem com 280×200 pixels. As 400 imagens assim obtidas estão em:

http://www.lps.usp.br/hae/apostila/feiCorCrop.zip (compactado) OU
https://drive.google.com/drive/folders/1WdR28ZopUOUG9Y7grJ7B0bvlezU3m-P3?usp=sharing (descompactado)

Nesse ZIP, também há 3 arquivos ".csv" (comma-separated values): *treino.csv*, *valida.csv* e *teste.csv*, com lista de nomes de arquivos seguida de classificação (0=masculino e 1=feminino). Por exemplo, o início de treino.csv:

```
019a.jpg;0
019b.jpg;0
072a.jpg;1
072b.jpg;1
(...)
```

Algumas imagens desse BD:



095a.jpg | 095b.jpg | 096a.jpg | 096b.jpg

Figura 1: Algumas imagens do BD feiCorCrop.

[Nota para mim: Neste exemplo, seria melhor fazer 5-fold cross validation.]

Nota: Se fosse realmente fazer validação, a rede com a maior acuracidade nos dados de validação deveria ser gravada no HD e depois utilizada para classificar os dados de teste. Não vamos fazer isso. Vamos simplesmente treinar a rede um certo número de épocas e testar o modelo obtido nos dados de validação e de teste.

A seguinte “célula de código Colab” baixa e descompacta esse BD no diretório local, se ainda não estiver lá.

```
1 url='http://www.lps.usp.br/hae/apostila/feiCorCrop.zip'
2 import os; nomeArq=os.path.split(url)[1]
3 if not os.path.exists(nomeArq):
4   print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
5   os.system("wget -nc -U 'Firefox/50.0' "+url)
6 else:
7   print("O arquivo",nomeArq,"ja existe no diretorio default",os.getcwd())
8 print("Descompactando arquivos novos de",nomeArq)
9 os.system("unzip -u "+nomeArq)
```

Programa 1: “Célula de código Colab” para baixar e descompactar BD.

O programa 2 abaixo utiliza rede “tipo LeNet” (sem data augmentation) e faz treino “do zero” para classificar imagens em masculino ou feminino. A taxa de acerto de validação/teste é entre 94%-96%, executando o treino 6 vezes (tabela 1). A taxa de acerto de treino é 100%, indicando que há *overfitting*. A perda para de diminuir nas últimas épocas, indicando que não adianta treinar durante mais tempo que a taxa de acerto não vai melhorar.

Exercício: Acrescente data augmentation no programa 2 para ver se é possível aumentar a taxa de acertos de teste/validação.

Exercício: Introduza outras alterações no programa 2 para aumentar a taxa de acertos de teste/validação.

Exercício: Utilize outros modelos de redes (como VGG, Inception e ResNet) e faça treino “do zero”, para verificar se é possível obter taxa de acerto substancialmente melhor do que 94-95%.

Tabela 1: Acuracidades obtidas em seis execuções do programa 2 (em %).

	exec #0	exec #1	exec #2	exec #3	exec #4	exec #5	média	desvio
treino	100	100	100	100	100	100	100	0,0
validação	93	97	97	94	97	94	95,3	1,9
teste	95	94	95	95	95	95	94,8	0,4

```
Epoch 1/50  - 0s - loss: 0.3977     - accuracy: 0.4900 - val_loss: 0.2444 - val_accuracy: 0.5500
Epoch 10/50 - 0s - loss: 0.0402     - accuracy: 0.9750 - val_loss: 0.0725 - val_accuracy: 0.9000
Epoch 20/50 - 0s - loss: 6.7161e-04 - accuracy: 1.0000 - val_loss: 0.0595 - val_accuracy: 0.9200
Epoch 30/50 - 0s - loss: 1.3730e-04 - accuracy: 1.0000 - val_loss: 0.0605 - val_accuracy: 0.9200
Epoch 40/50 - 0s - loss: 1.3431e-04 - accuracy: 1.0000 - val_loss: 0.0587 - val_accuracy: 0.9200
Epoch 46/50 - 0s - loss: 1.9087e-04 - accuracy: 1.0000 - val_loss: 0.0593 - val_accuracy: 0.9200
Epoch 47/50 - 0s - loss: 6.1102e-04 - accuracy: 1.0000 - val_loss: 0.0587 - val_accuracy: 0.9300
Epoch 48/50 - 0s - loss: 9.6069e-04 - accuracy: 1.0000 - val_loss: 0.0605 - val_accuracy: 0.9300
Epoch 49/50 - 0s - loss: 9.6738e-04 - accuracy: 1.0000 - val_loss: 0.0573 - val_accuracy: 0.9400
Epoch 50/50 - 0s - loss: 0.0015     - accuracy: 1.0000 - val_loss: 0.0582 - val_accuracy: 0.9400
Training loss: [0.0014237392460927367, 1.0]
Validation loss: [0.05823998898267746, 0.9399999976158142]
Test loss: [0.04361637681722641, 0.949999988079071]
```

Figura 2: Saída da execução #5 do programa 2. A rede parou de melhorar nas últimas épocas.

```python
#fei_LeNet.py
#Mean training accuracy: 100%. Mean validation accuracy: 88.7%. Mean test accuracy: 91.2%
import os; os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
import tensorflow.keras as keras
import cv2; import numpy as np; import os
import tensorflow.keras as keras; import keras.backend as K;
from keras import optimizers, callbacks, regularizers;
from keras.regularizers import l2;
from keras.models import Sequential;
from keras.layers import Dropout, Conv2D, MaxPooling2D, Dense, Flatten;
from inspect import currentframe, getframeinfo

def leCsv(nomeDir, nomeArq, nl=0, nc=0, menosmais=False):
  #nomeDir = Diretorio onde estao treino.csv, teste.csv e imagens nnna.jpg e nnnb.jpg.
  #Ex: nomeDir = "/home/hae/haebase/fei/feiFrontCor"
  #Imagens sao redimensionadas para nlXnc (se diferentes de zero).
  st=os.path.join(nomeDir,nomeArq); arq=open(st,"rt")
  lines=arq.readlines(); arq.close(); n=len(lines)

  linhas_separadas=[]
  for linha in lines:
    linha=linha.strip('\n'); linha=linha.split(';'); linhas_separadas.append(linha);
  ay=np.empty((n),dtype='float32'); ax=np.empty((n,nl,nc,3),dtype='float32');
  for i in range(len(linhas_separadas)):
    linha=linhas_separadas[i];
    t=cv2.imread(os.path.join(nomeDir,linha[0]),1);
    if nl>0 and nc>0:
      t=cv2.resize(t,(nc,nl),interpolation=cv2.INTER_AREA);
    ax[i]=np.float32(t)/255.0; #Entre 0 e 1
    if menosmais:
      ax[i]=ax[i]-0.5 #-0.5 a +0.5
    ay[i]=np.float32(linha[1]); #0=m ou 1=f
  return ax, ay;

#<<<<<<<<<<<<<<<<<<<< main <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
#Original: 280x200, redimensionado: 112x80
nl=112; nc=80
diretorioBd="."
ax, ay = leCsv(diretorioBd,"treino.csv", nl=nl, nc=nc, menosmais=True); #200 imagens
qx, qy = leCsv(diretorioBd,"teste.csv",  nl=nl, nc=nc, menosmais=True);  #100 imagens
vx, vy = leCsv(diretorioBd,"valida.csv", nl=nl, nc=nc, menosmais=True);  #100 imagens
input_shape = (nl,nc,3); batch_size = 10; epochs = 50;

model = Sequential();
model.add(Conv2D(30, kernel_size=(5,5), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2))) #56x40
model.add(Conv2D(40, kernel_size=(5,5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2))) #28x20
model.add(Conv2D(50, kernel_size=(5,5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2))) #14x10
model.add(Conv2D(60, kernel_size=(5,5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2))) #7x5
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dense(1, activation='linear'))

#from tensorflow.keras.utils import plot_model
#plot_model(model, to_file='ep2g.png', show_shapes=True)
#model.summary()

opt=optimizers.Adam();
model.compile(optimizer=opt, loss='mse', metrics=['accuracy'])
model.fit(ax, ay, batch_size=batch_size,
  epochs=epochs, verbose=2, validation_data=(vx,vy))

score = model.evaluate(ax, ay, verbose=0)
print('Training loss:', score)
score = model.evaluate(vx, vy, verbose=0)
print('Validation loss:', score)
score = model.evaluate(qx, qy, verbose=0)
print('Test loss:', score)
```

Programa 2: Rede “tipo LeNet” sem data augmentation para classificar feiCorCrop em M/F. Chega à acuracidade de aproximadamente 95%.
https://colab.research.google.com/drive/1fY31WECVLztkxXB44COKjqesuA5BZ1u3?usp=sharing

2. Transfer learning para classificação M/F

Aparentemente, é difícil aumentar a acuracidade da classificação de rostos em M/F substancialmente acima de 94-96%, pois a quantidade de imagens de treino é pequena (200 imagens).

Num caso como este, é possível usar “transfer learning” para começar o treino a partir de uma rede com taxa de acerto mais alta, fazer a rede convergir mais rapidamente e atingir uma acuracidade final maior (figura 3). Vamos usar algumas ideias de [https://machinelearningmastery.com/transfer-learning-for-deep-learning/] para explicar este conceito.

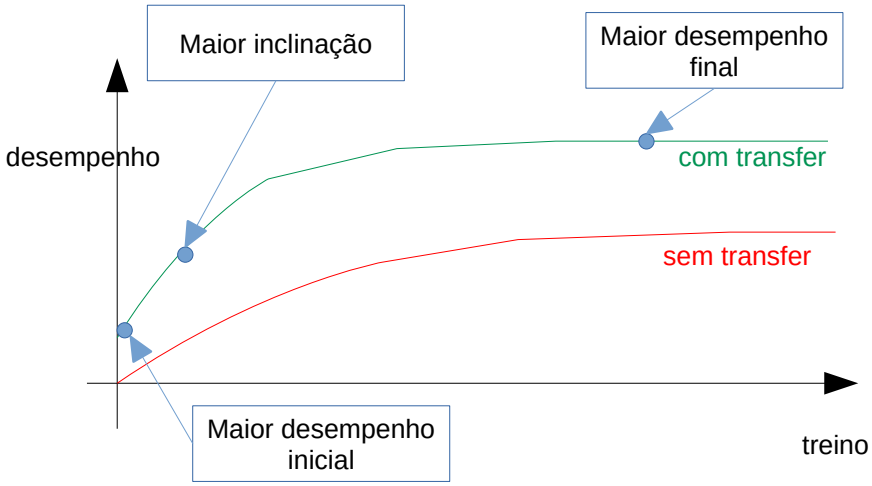


Figura 3: Utilizando “transfer learning”, o desempenho é maior já no início do treino, cresce mais rapidamente e atinge valor final maior do que sem usar essa técnica.

“Transfer learning” consiste em reutilizar um modelo pré-treinado como ponto inicial para resolver um novo problema. Isto é, utilizar uma rede neural treinada num outro conjunto de dados, normalmente maior, para resolver um novo problema.

Por exemplo, as “redes convolucionais modernas” (VGG, Inception, ResNet, EfficientNet, etc) que estudamos na apostila “cifar-ead” (para classificar as imagens de ImageNet em 1000 categorias) podem ser utilizadas como ponto de partida para resolver a classificação de rostos em M/F. A explicação intuitiva é que essas redes, para poder classificar as imagens, aprenderam a extrair atributos visuais úteis. Note que as faces humanas não estão incluídas nas 1000 classes de ImageNet:

https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/

Site [https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/] resume as quatro diferentes técnicas de “transfer learning”:

1. *Classificador:* O modelo pré-treinado é usado para classificar diretamente novas imagens. Por exemplo, digamos que queremos classificar as imagens em “gato”, “cachorro” e “nda”. Para isso, poderia usar qualquer rede treinada em ImageNet que classifica imagens em 1000 categorias. Se a classificação do modelo for alguma das raças de gato (281 tabby cat até 285 Egyptian cat), a classificação seria “gato”. Se a resposta for alguma das raças de cachorro (151 Chihuahua até 268 Mexican hairless), a resposta seria “cachorro”. Seria “nda” se as alternativas anteriores forem falsas.
2. *Extrator de atributos “stand-alone”:* O modelo pré-treinado (normalmente sem as últimas camadas) é utilizado para extrair atributos relevantes da imagem. Estes atributos são usados por classificadores que não precisam ser necessariamente redes neurais. Poderia usar, por exemplo, árvore de decisão, vizinho mais próximo, boosting, classificador de Bayes, SVM, etc.
3. *Extrator de atributos integrado (chamado de simplesmente **transfer learning**):* O modelo pré-treinado (normalmente sem as últimas camadas) é integrado a um novo modelo acrescentando novas camadas (normalmente de topo). As camadas do modelo pré-treinado são congeladas durante o treino. Isto é, somente as novas camadas (normalmente as superiores) são treinadas para resolver o novo problema.
4. *Inicializador de pesos (**fine tuning**):* O modelo pré-treinado (normalmente sem as últimas camadas) é integrado ao novo modelo, e tanto as camadas do modelo pré-treinado quanto as novas camadas são treinadas para se adequar ao novo problema. Neste caso, costuma-se utilizar *learning rate* bem pequena, para não “estragar” os pesos do modelo pré-treinado.

As mais interessantes são as técnicas (3) e (4). Muitas vezes, as técnicas (3) e (4) aparecem integradas. Primeiro, utiliza-se a técnica (3) com learning rate grande para treinar rapidamente os pesos das novas camadas. Depois, utiliza-se a técnica (4) com learning rate bem pequena para fazer calibração fina dos pesos da rede pré-treinada.

Vamos utilizar esta ideia (usar as técnicas (3) e (4) de forma integrada, uma depois da outra) usando a rede ResNet50, para classificar rostos em M/F.

Exercício: Testar a técnica 2 (extrator de atributos stand-alone) no Cifar-10. Use (por exemplo) ResNet50 e extraia os atributos. Depois, use esses atributos para classificar as imagens em 2 classes com (por exemplo) vizinho mais próximo.

[Retirei a seção 4: Transfer learning de VGG para encurtar a aula. Esta seção está em transfer-ead-velho1.odt]

3. Transfer learning de ResNet50 para classificação M/F

Vamos fazer transfer learning, usando ResNet50 como modelo-base, para classificar rostos em M/F (programa 3). Antes de executar este programa, deve executar primeiro "célula de código Colab" do programa 1 para baixar e descompactar o BD localmente.

No programa principal, você deve inicializar variável *diretorioBD* (linha 39) com o diretório onde está BD. Estamos redimensionando imagens para 224×224, pois é o padrão de ResNet50 (linha 38).

As linhas 16-42 do programa 3 fazem leitura das imagens. As imagens lidas devem ser tratadas pela função *preprocess_input* (linha 30), onde cada modelo de rede possui a sua própria função *preprocess_input*. Por exemplo, não se pode usar *preprocess_input* de VGG16 em ResNet50 ou vice-versa. Dependendo da rede, essa função pode colocar os níveis de cinza em intervalos diferentes.

A linha 48 lê a rede ResNet50 com os pesos treinados para ImageNet, sem as camadas average pooling e densas do topo da rede (figura 4). Na saída da rede ResNet50 sem topo, vamos acrescentar as camadas de global average pooling, flatten e uma camada densa com 2 neurônios, para ficar igual à arquitetura original de ResNet50.

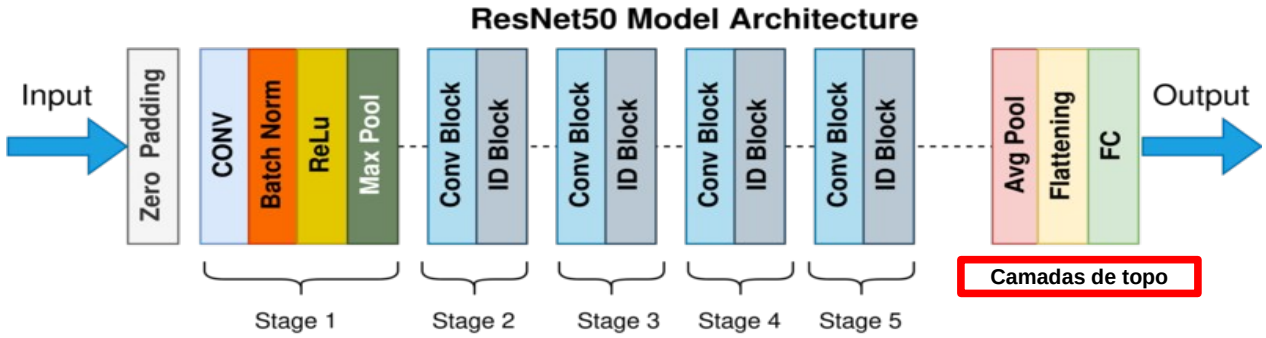


Figura 4: Estrutura de ResNet50 destacando as camadas de topo.
https://commons.wikimedia.org/wiki/File:ResNet50.png

As linhas 57-62 treinam somente as duas novas camadas densas do topo durante 40 épocas usando learning rate grande (1e-4). As linhas 68-72 treinam todas as camadas durante 40 épocas usando learning rate pequena (1e-7).

As acuracidades obtidas no final do programa estão na tabela 2. O treino foi repetido 6 vezes, obtendo acuracidades médias de validação e teste respectivamente 99,00% e 98,50%. Isto é bem mais do que 95,4% e 94,8% que tínhamos obtido treinando rede "tipo LeNet" do zero.

Implementando "ensemble" das 6 redes, obtive as acuracidades de validação 100% e teste 99%.

```python
#resnet_transf.py
#Faz transfer learning usando ResNet50.
#Execute 6 vezes mudando o nomeprog de resnet_transf0 ate resnet_transf5
#https://medium.com/abraia/first-steps-with-transfer-learning-for-custom-image-classification-with-keras-b941601fcad5
import os; os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
import numpy as np;
import tensorflow.keras as keras; import keras.backend as K;
from tensorflow.keras import optimizers, callbacks, regularizers;
from tensorflow.keras.regularizers import l2;
from tensorflow.keras.models import Sequential, Model;
from tensorflow.keras.layers import Dropout, Conv2D, MaxPooling2D, Dense, Flatten, GlobalAveragePooling2D;
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image

def leCsv(nomeDir, nomeArq, nl=0, nc=0):
  st=os.path.join(nomeDir,nomeArq);
  arq=open(st,"rt"); lines=arq.readlines(); arq.close(); n=len(lines)

  linhas_separadas=[]
  for linha in lines:
    linha=linha.strip('\n'); linha=linha.split(';'); linhas_separadas.append(linha);

  ay=np.empty((n),dtype='float32'); ax=np.empty((n,nl,nc,3),dtype='float32');
  for i in range(len(linhas_separadas)):
    linha=linhas_separadas[i];
    img_path=os.path.join(nomeDir,linha[0])
    t = image.load_img(img_path, target_size=(nl,nc))
    x = image.img_to_array(t)
    x = np.expand_dims(x, axis=0)
    ax[i] = preprocess_input(x)
    ay[i] = np.float32(linha[1]); #0=m ou 1=f
  return ax, ay;

#<<<<<<<<<<<<<<<<<<<<<<< main <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
nomeprog="resnet_transf0"
#Original: 280x200, redimensionado: 224x224
num_classes=2; nl=224; nc=224
diretorioBd="."
ax, ay = leCsv(diretorioBd,"treino.csv", nl=nl, nc=nc); #200 imagens
qx, qy = leCsv(diretorioBd,"teste.csv", nl=nl, nc=nc);  #100 imagens
vx, vy = leCsv(diretorioBd,"valida.csv", nl=nl, nc=nc);  #100 imagens
ay = keras.utils.to_categorical(ay, num_classes)
qy = keras.utils.to_categorical(qy, num_classes)
vy = keras.utils.to_categorical(vy, num_classes)

input_shape = (nl,nc,3); batch_size = 10;
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
# base_model = ResNet50(weights=None, include_top=False, input_shape=input_shape)
# base_model.summary()
x = base_model.output
x = GlobalAveragePooling2D()(x) #
x = Flatten()(x)
predictions = Dense(num_classes, activation="softmax")(x)
model = Model(inputs=base_model.input, outputs=predictions)

#Nao permite treinar base_model. So as camadas densas sao treinadas:
for layer in base_model.layers: layer.trainable = False
#Treina com learning rate grande
otimizador=keras.optimizers.Adam(learning_rate=1e-4)
model.compile(otimizador, loss='categorical_crossentropy', metrics =['accuracy'])
model.fit(ax, ay, batch_size=batch_size, epochs=40, verbose=2, validation_data=(vx,vy))

score = model.evaluate(ax, ay, verbose=0); print('Training loss:', score)
score = model.evaluate(vx, vy, verbose=0); print('Validation loss:', score)
score = model.evaluate(qx, qy, verbose=0); print('Test loss:', score)

#Libera todos layers do model (incluindo modelo-base) para treinar:
for layer in model.layers: layer.trainable = True
#Treina com learning rate pequena todas as camadas
model.learning_rate=1e-7
model.fit(ax, ay, batch_size=batch_size, epochs=40, verbose=2, validation_data=(vx,vy))

score = model.evaluate(ax, ay, verbose=0); print('Training loss:', score)
score = model.evaluate(vx, vy, verbose=0); print('Validation loss:', score)
score = model.evaluate(qx, qy, verbose=0); print('Test loss:', score)
model.save(nomeprog+".h5")
```

Programa 3: Transfer learning para classificar rostos em M/F, usando ResNet50 como modelo-base.
https://colab.research.google.com/drive/1NfIwVlcCjED5bbETMHUJRbQIYqv5zaLy?usp=sharing

```
Model: "resnet50"
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_33 (InputLayer)           (None, 224, 224, 3)  0
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_33[0][0]
conv1_conv (Conv2D)             (None, 112, 112, 64) 9472        conv1_pad[0][0]
conv1_bn (BatchNormalization)   (None, 112, 112, 64) 256         conv1_conv[0][0]
conv1_relu (Activation)         (None, 112, 112, 64) 0           conv1_bn[0][0]
(...)
conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)    1049088     conv5_block2_out[0][0]
conv5_block3_1_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation (None, 7, 7, 512)    0           conv5_block3_1_bn[0][0]
conv5_block3_2_conv (Conv2D)    (None, 7, 7, 512)    2359808     conv5_block3_1_relu[0][0]
conv5_block3_2_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation (None, 7, 7, 512)    0           conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     conv5_block3_2_relu[0][0]
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block3_3_conv[0][0]
conv5_block3_add (Add)          (None, 7, 7, 2048)   0           conv5_block2_out[0][0]
                                                                 conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0           conv5_block3_add[0][0]
==================================================================================================
```

Figura 5: Modelo ResNet50 sem as camadas densas do final da rede.

Tabela 2: Acuracidades obtidas nas 6 execuções de transfer learning usando ResNet50 como base (em %)

	exec #0	exec #1	exec #2	exec #3	exec #4	exec #5	média	desvio
treino	99	99	99	99	99	99,5	99,1	0,2
validação	100	99	99	98	99	99	99,0	0,6
teste	99	98	99	97	99	99	98,5	0,8

Obtivemos acuracidades médias de validação e teste de 99,0% e 98,5%, muito maiores que 95,4% e 94,8% que tínhamos obtido com rede “tipo LeNet”.

Fazendo ensemble das 6 redes, obtive acuracidade maior que as médias:

```
Training accuracy: 99,0%
Validation accuracy: 100,0%
Test accuracy: 99,0%
```

É impressionante que o modelo cometeu um único erro ao classificar 200 faces (validação+teste) em masculino/feminino!

Exercício: Os modelos obtidos nas execuções de 0 a 5 foram gravadas como resnet_transf.zip e estão disponíveis em:

https://drive.google.com/file/d/1-8CP3rZu8ha8v2aeVKUyU2kkQYJTvRD2/view?usp=sharing

Implemente "ensemble" dessas 6 redes. Isto é, use esses 6 modelos para fazer 6 predições para cada imagem e calcule a média das 6 probabilidades de pertencer às classes M e F. A "ensemble" deve classificar a imagem na classe com a maior probabilidade média. Como já disse, fazendo isso, obtive acuracidade treino=99%, validação=100% e teste=99%.

Solução privada em https://colab.research.google.com/drive/1v33YzVutsLgnqahIgw1UTNexz3X158FR?usp=sharing

Código para baixar e descompactar as redes treinadas:

```
import os; import gdown
nomeArq="resnet_transf.zip"
if not os.path.exists(nomeArq):
  !gdown --id 1-8CP3rZu8ha8v2aeVKUyU2kkQYJTvRD2
!unzip -u resnet_transf
```

Exercício: Você acha que a taxa de acerto irá melhorar usando TTA? Justifique.

Solução privada em **https://colab.research.google.com/drive/19__JBdZya1i31n9CjRdTT1Lo33404Q78**

4. Classificação de rostos em M/F usando rede “ResNet50” treinada “do zero”

O que acontece se no programa 3 não carregar os pesos do ImageNet? O programa 4 é igual ao programa 3. Somente troco os comentários das linhas 47 e 48, fazendo com que o programa não carregue os pesos de ImageNet, mas inicialize com valores aleatórios.

```
1  #resnet_zero.py
2  #Faz transfer learning usando ResNet50.
3  #https://medium.com/abraia/first-steps-with-transfer-learning-for-custom-image-classification-with-keras-b941601fcad5
4  import os; os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
5  os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
6  import numpy as np;
7  import tensorflow.keras as keras; import keras.backend as K;
8  from tensorflow.keras import optimizers, callbacks, regularizers;
9  from tensorflow.keras.regularizers import l2;
10 from tensorflow.keras.models import Sequential, Model;
11 from tensorflow.keras.layers import Dropout, Conv2D, MaxPooling2D, Dense, Flatten;
12 from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions
13 from tensorflow.keras.preprocessing import image
14
15 def leCsv(nomeDir, nomeArq, nl=0, nc=0):
16   st=os.path.join(nomeDir,nomeArq);
17   arq=open(st,"rt"); lines=arq.readlines(); arq.close(); n=len(lines)
18
19   linhas_separadas=[]
20   for linha in lines:
21     linha=linha.strip('\n'); linha=linha.split(';'); linhas_separadas.append(linha);
22
23   ay=np.empty((n),dtype='float32'); ax=np.empty((n,nl,nc,3),dtype='float32');
24   for i in range(len(linhas_separadas)):
25     linha=linhas_separadas[i];
26     img_path=os.path.join(nomeDir,linha[0])
27     t = image.load_img(img_path, target_size=(nl,nc))
28     x = image.img_to_array(t)
29     x = np.expand_dims(x, axis=0)
30     ax[i] = preprocess_input(x)
31     ay[i] = np.float32(linha[1]); #0=m ou 1=f
32   return ax, ay;
33
34 #<<<<<<<<<<<<<<<<<<<<<<< main <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
35 nomeprog="resnet_zero0"
36 #Original: 280x200, redimensionado: 224x224
37 num_classes=2; nl=224; nc=224
38 diretorioBd="."
39 ax, ay = leCsv(diretorioBd,"treino.csv", nl=nl, nc=nc); #200 imagens
40 qx, qy = leCsv(diretorioBd,"teste.csv", nl=nl, nc=nc);  #100 imagens
41 vx, vy = leCsv(diretorioBd,"valida.csv", nl=nl, nc=nc);  #100 imagens
42 ay = keras.utils.to_categorical(ay, num_classes)
43 qy = keras.utils.to_categorical(qy, num_classes)
44 vy = keras.utils.to_categorical(vy, num_classes)
45
46 input_shape = (nl,nc,3); batch_size = 10;
47 #base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
48 base_model = ResNet50(weights=None, include_top=False, input_shape=input_shape)
49 # base_model.summary()
50 x = base_model.output
51 x = GlobalAveragePooling2D()(x) #
52 x = Flatten()(x)
53 predictions = Dense(num_classes, activation="softmax")(x)
54 model = Model(inputs=base_model.input, outputs=predictions)
55
56 #Nao permite treinar base_model. So as camadas densas sao treinadas:
57 for layer in base_model.layers: layer.trainable = False
58 #Treina com learning rate grande
59 otimizador=keras.optimizers.Adam(learning_rate=1e-4)
60 model.compile(otimizador, loss='categorical_crossentropy', metrics =['accuracy'])
61 model.fit(ax, ay, batch_size=batch_size, epochs=40, verbose=2, validation_data=(vx,vy))
62
63 score = model.evaluate(ax, ay, verbose=0); print('Training loss:', score)
64 score = model.evaluate(vx, vy, verbose=0); print('Validation loss:', score)
65 score = model.evaluate(qx, qy, verbose=0); print('Test loss:', score)
66
67 #Libera todos layers do model (incluindo modelo-base) para treinar:
68 for layer in model.layers: layer.trainable = True
69 #Treina com learning rate pequena todas as camadas
70 model.learning_rate=1e-7
71 model.fit(ax, ay, batch_size=batch_size, epochs=40, verbose=2, validation_data=(vx,vy))
72
73 score = model.evaluate(ax, ay, verbose=0); print('Training loss:', score)
74 score = model.evaluate(vx, vy, verbose=0); print('Validation loss:', score)
75 score = model.evaluate(qx, qy, verbose=0); print('Test loss:', score)
76 model.save(nomeprog+".h5")
```

Programa 4: Classificar rostos em M/F, usando ResNet50 com pesos aleatórios e o mesmo número de épocas que programa 3. https://colab.research.google.com/drive/1T_GP-QKuwmJOUjw4bH9KkBLPLYpgvSOu?usp=sharing

Executei programa 4 (ResNet com pesos inicializados aleatoriamente) uma única vez usando os mesmos parâmetros que programa 3 (transfer learning) e obtive:

Tabela 3: Acuracidades obtidas ao classificar M/F sem transfer learning.

	exec #0
treino	84%
validação	88%
teste	85%

Estes resultados são bem piores do que aqueles obtidos usando rede “tipo LeNet”, onde obtivemos aproximadamente 95% de acuracidade de validação e teste. Provavelmente, é necessário treinar durante mais épocas e acertar melhor os parâmetros para melhorar a acuracidade...

II. Classificar Cifar-10 fazendo transfer learning de EfficientNet

Adaptado de:
https://www.kaggle.com/code/nikhilpandey360/transfer-learning-using-xception

A maior taxa de acerto de Cifar-10 que conseguimos na apostila “cifar-ead” foi de 93,4% usando TTA (test-time augmentation). Quase certamente a taxa de acerto aumentaria um pouco mais usando ensemble, mas não fizemos pois o processamento demoraria excessivamente.

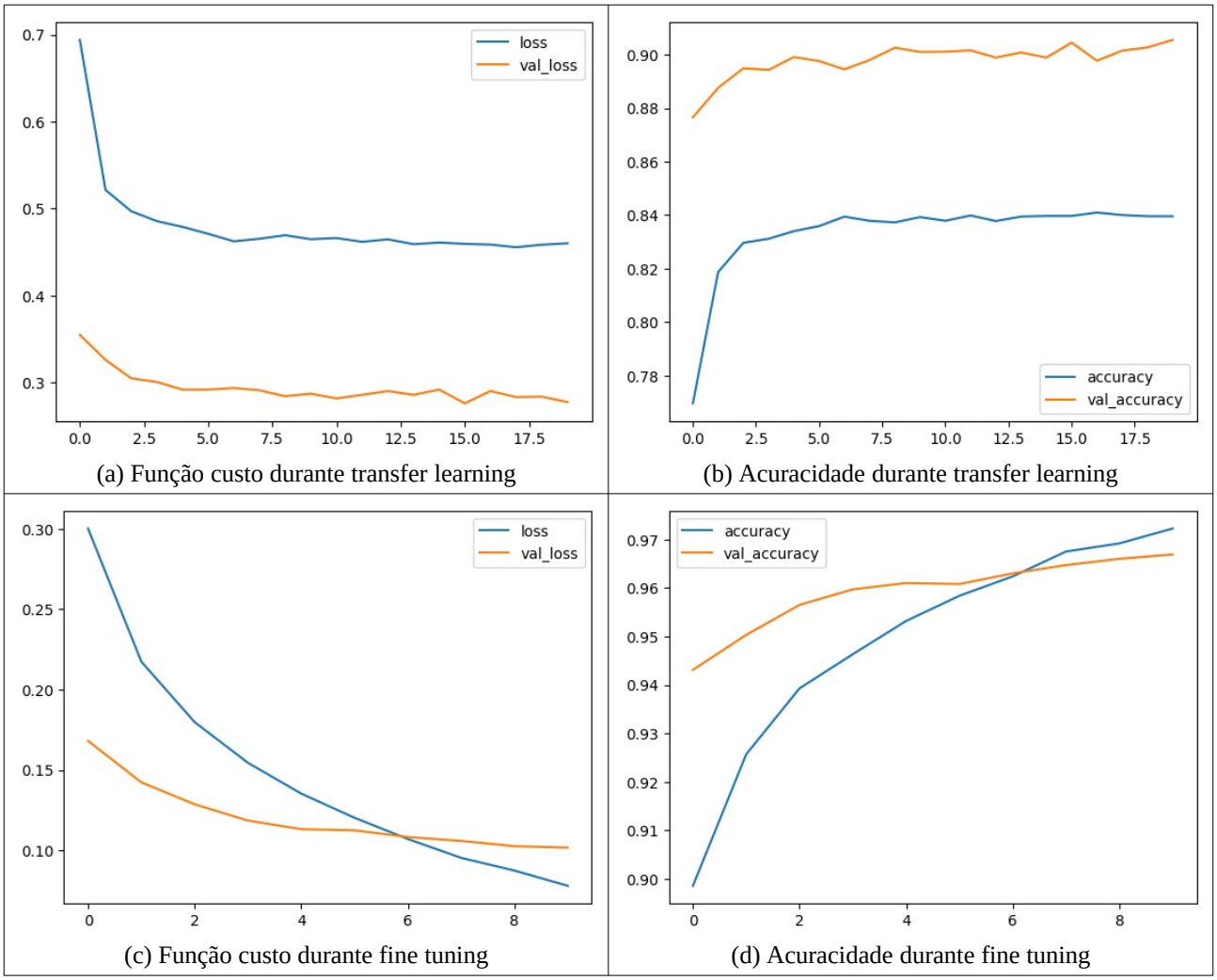
Nesta seção, veremos que usando transfer learning consegue-se taxa de acerto de 96,4% gastando pouco tempo de treino. Quase certamente, a acuracidade seria ainda maior usando TTA e/ou ensemble.

EfficientNet-B0 possui resolução de entrada 224×224×3. As imagens de Cifar-10 possuem resolução 32×32×3. Assim, as imagens de Cifar-10 serão redimensionadas para 224×224×3 para poder fazer transfer learning.

Vamos fazer data augmentation:

```
[RandomFlip("horizontal"), RandomRotation(0.1), RandomZoom(0.1)]
```

Executaremos transfer learning congelando as camadas de EfficientNet-B0 durante 20 épocas. Depois, faremos fine tuning, descongelando todas as camadas durante 10 épocas.



(a) Função custo durante transfer learning

(b) Acuracidade durante transfer learning

(c) Função custo durante fine tuning

(d) Acuracidade durante fine tuning

Figura 6: Histórico de treino de transfer learning (a-b) e fine tuning (c-d).

Tabela 4: Acuracidades classificando Cifar-10 fazendo transfer learning do EfficientNetB0.

	Acurac. treino	Acurac. validação	Acurac. teste
Após transfer learning	0.8396	0.9055	0.9000
Após fine tuning	0.9722	0.9669	0.9637

Ou seja, atingimos 96,37% de taxa de acerto, a maior taxa de acerto obtida até aqui (que tinha sido 93,4%).

Cada época de transfer learning demorou 92s e cada época de fine tuning demorou 300s.

Na matriz de confusão abaixo, podemos observar que muitas vezes gato é confundido com cachorro; automóvel com caminhão; e avião com navio. Isto era o esperado.

O programa plota matriz de confusão, destacado em amarelo no código.

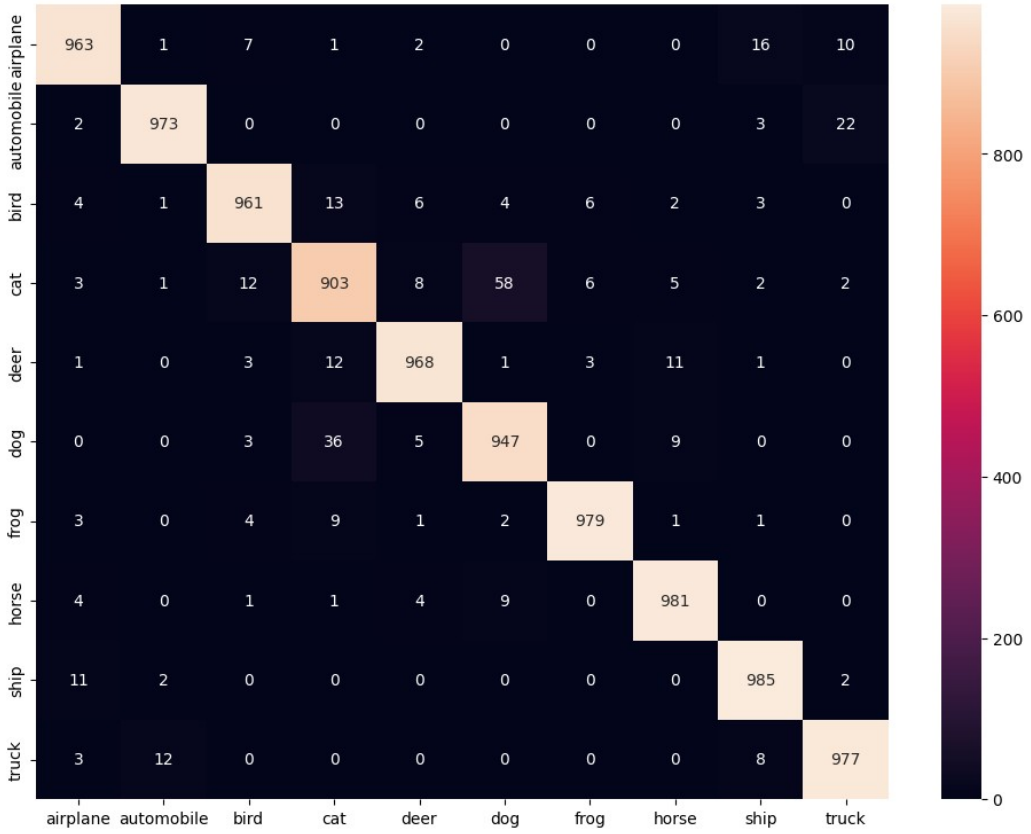


Figura 7: Matriz de confusão com os resultados do programa 5.

```
"""
~/deep/algpi/transf/efficientnet_transf_cifar10.py
Original file: https://colab.research.google.com/drive/1bD_ckH-KiPL_lgheQ7SDQOv6x4QO_EM0
Baseado em: https://www.kaggle.com/code/nikhilpandey360/transfer-learning-using-xception
"""
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np; import pandas as pd
import warnings; warnings.filterwarnings("ignore")
import tensorflow as tf; print(tf.__version__)
from keras import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD,Adam
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import Input, Flatten, Dense, BatchNormalization, Activation,\
  Dropout, GlobalAveragePooling2D, MaxPooling2D, RandomFlip, RandomZoom, RandomRotation
from keras.datasets import cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train=to_categorical(y_train); y_test=to_categorical(y_test)
print((x_train.shape, y_train.shape)); print((x_test.shape, y_test.shape))

base_model = EfficientNetB0(include_top=False, weights='imagenet', input_shape=(224,224,3), classes=y_train.shape[1])
base_model.trainable = False
#base_model.summary()

data_augmentation = Sequential(
    [RandomFlip("horizontal"), RandomRotation(0.1), RandomZoom(0.1)]
)
"""Consider the image resolution that the imagenet was trained on.
The original image resolution of CIFAR-10 is 32x32, which is too low for EfficientNetB0 (min. 71x71)
O tamanho padrao de entrada desta rede e' 224x224
"""
inputs = tf.keras.Input(shape=(32, 32, 3))
x = tf.keras.layers.Lambda(lambda image: tf.image.resize(image, (224,224)))(inputs)
# x = tf.keras.layers.Resizing(224,224)(inputs)
x = data_augmentation(x)
x = tf.keras.applications.efficientnet.preprocess_input(x)
x = base_model(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(10, activation=('softmax'))(x)
model = tf.keras.Model(inputs, outputs)
model.summary()

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
epochs = 20
history = model.fit(x_train, y_train, validation_split=0.2, epochs=epochs, verbose=1)

def plot_history(history):
    history_frame = pd.DataFrame(history.history)
    history_frame.loc[:, ['loss', 'val_loss']].plot()
    history_frame.loc[:, ['accuracy', 'val_accuracy']].plot()
    return
plot_history(history)

test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy", test_acc); print("Test loss", test_loss)

"""# Fine Tuning"""
base_model.trainable = True
model.compile(optimizer=tf.keras.optimizers.Adam(1e-5), loss='categorical_crossentropy', metrics=['accuracy'])
epochs = 10
history = model.fit(x_train, y_train, validation_split=0.2, epochs=epochs, verbose=1)

test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy", test_acc); print("Test loss", test_loss)
plot_history(history)

"""# Confusion Matrix"""
class_names=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
predictions=model.predict(x_test)
y_pred_classes = np.argmax(predictions, axis=1)
y_true = np.argmax(y_test, axis=1)
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(12, 9))
c = sns.heatmap(confusion_mtx, annot=True, fmt='g')
c.set(xticklabels=class_names, yticklabels=class_names)
```

Programa 5: Classificar Cifar-10, fazendo transfer learning de EfficientNet-B0.
https://colab.research.google.com/drive/1bD_ckH-KiPL_lgheQ7SDQOv6x4QO_EM0#scrollTo=bCEWvAQqiNLF

Exercício: Modifique o programa 5 para classificar fashion-mnist. Para acelerar o processamento, execute somente 10 épocas para fazer "ajuste grosso" e 5 épocas para "ajuste fino". Qual foi a taxa de acerto fazendo transfer learning? A taxa de acerto foi maior do que aquela obtida sem fazer transfer learning? Nota: obtivemos 94% *sem* fazer transfer learning e o recorde é 96,91%.
Solução privada em: https://colab.research.google.com/drive/1Ubj8KP0bZJRPliDq70i5qUhEbZu7Np4Y#scrollTo=aC4JTeDXiNLH

[PSI3472-2023. Aulas 7/8. Lição de casa extra 1 (vale +2 pontos).] No programa 5, conseguimos atingir taxa de acerto de teste 96,37% ao classificar Cifar-10 fazendo transfer learning de Efficient-Net-B0. Melhore o programa 5 e chegar a acuracidade de teste maior que 97% ao classificar Cifar-10. *Nota:* Não resolvi este exercício. Não sei se é possível resolvê-la.
Possíveis técnicas:
a) Usar data augmentation.
b) Mudar os parâmetros do programa.
c) Usar regularização L1 ou L2 nas camadas de topo.
d) Usar função callback para diminuir learning rate quando o desempenho da rede para de melhorar.
e) Gravar e usar o modelo que atinge a maior taxa de acerto de validação.
f) Usar TTA (test time augmentation).
g) Usar ensemble.

[PSI3472-2023. Aulas 7/8. Lição de casa.] Copio abaixo o enunciado de EP de PSI3472-2021, adaptando-o ser feita como um exercício “lição de casa”.

http://www.lps.usp.br/hae/psi3472/ep-2021/index.html

Solução privada em: /home/hae/haepi/algpi/ep/cat-dog-pos2021

O objetivo deste exercício é identificar se numa imagem aparece gato ou cachorro. Para isso, construa um classificador usando “transfer learning”.

No site https://www.kaggle.com/rafaelrlima/cat-and-dog-clean-dataset há um banco de dado com 10.000 imagens, distribuidas em:

Treino: 4.000 imagens de gatos e 4.000 imagens de cachorros.

Teste: 1.000 imagens de gatos e 1.000 imagens de cachorros.

Deixei uma cópia desse BD em:

www.lps.usp.br/hae/psi3472/ep-2021/cat_dog_clean.zip

A figura abaixo mostra 10 primeiras imagens “em ordem alfabética” de gatos e cachorros para o treino.

Figura 1: 10 primeiras imagens de treino de gatos e cachorros em ordem alfabética.

Faça dois programas Python3/Keras: treino_transf.py e teste_transf.py. O programa treino_transf.py deve fazer o treino usando “transfer learning”. Você deve obrigatoriamente utilizar algum modelo pré-treinado e adaptá-lo para o problema dado. Você deve gravar o arquivo transf.h5 no diretório default (ou no Google Drive) com a rede treinada. Se quiser, pode usar “data augmentation” e outras técnicas para diminuir a taxa de erro. Depois do treino, o programa teste_transf.py deve carregar transf.h5 e classificar as imagens de teste, obtendo a taxa de erro de teste. Também deve imprimir as 10 primeiras imagens de gatos e 10 primeiras imagens de cachorros classificadas incorretamente.

Sem fazer “data augmentation” e sem fazer muito esforço para diminuir erro, obtive taxa de erro de 1,25%. A figura abaixo mostra as 10 primeiras imagens classificadas incorretamente.

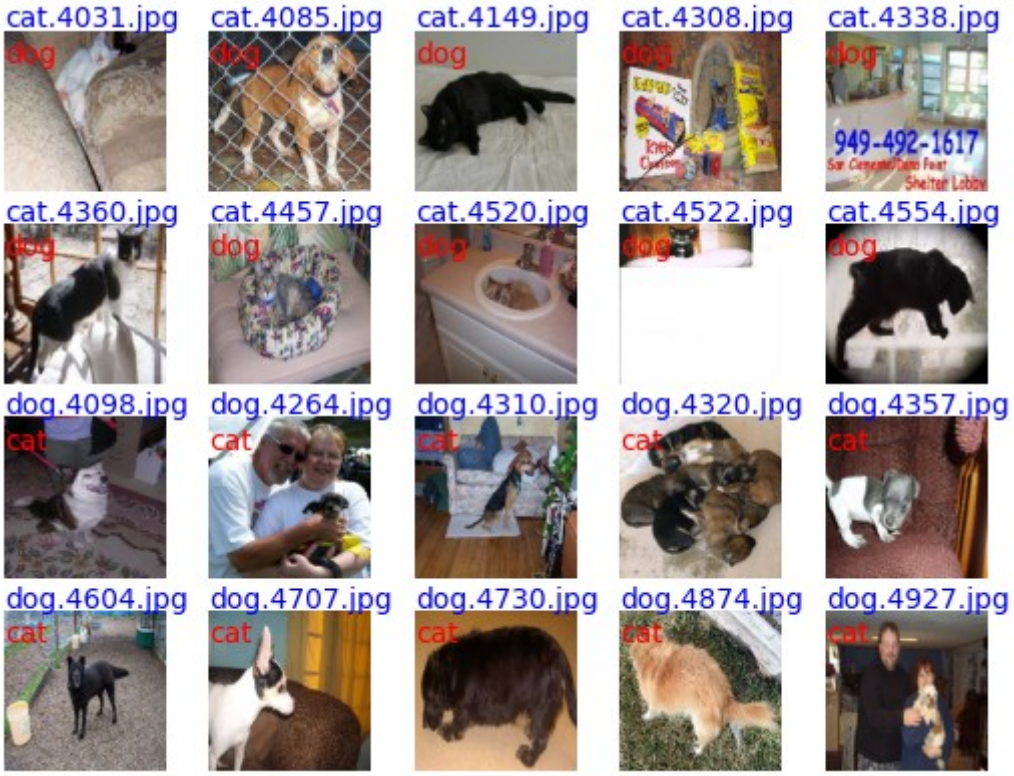


Figura 3: 10 primeiras imagens de teste classificadas incorretamente usando “transfer learning”.

Ajudas:

1) A célula abaixo baixa e descompacta BD.

```
url='http://www.lps.usp.br/hae/psi3472/ep-2021/cat_dog_clean.zip'
import os; nomeArq=os.path.split(url)[1]
if not os.path.exists(nomeArq):
  print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
  os.system("wget -nc -U 'Firefox/50.0' "+url)
else:
  print("O arquivo",nomeArq,"ja existe no diretorio default",os.getcwd())
print("Descompactando arquivos novos de",nomeArq)
os.system("unzip -u "+nomeArq)
```

2) O comando abaixo:

```
$ import glob
$ cats=glob.glob(nomeDirCat+"/*.jpg"); cats.sort()
```

Cria a lista ordenada “cats” com os nomes de todos os arquivos com extensão “.jpg” do diretório *nomeDirCat*.

3) Se estourar o limite de RAM do Google Colab, você pode:
a) Usar *flow_from_directory* (veja apostila cifar-ead) ou uma função semelhante que não carrega todo o banco de dados na memória (mas fica carregando do HD quando necessário).
b) Carregar imagens com resolução menor do que a resolução padrão das redes pré-treinadas. Por exemplo, para ResNet50 ou EfficientNetB0, usar imagens 112×112 em vez de 224×224 .
c) Não usar todas as imagens para fazer treino. Por exemplo, usar 2000+2000 imagens em vez de usar 4000+4000.

[PSI3472-2023. Aula 7. Fim.]